Data Streams: *Probabilistic Counting*

Universitat Pompeu Fabra Barcelona

Mining Massive Datasets

Materials provided by Prof. Carlos Castillo — https://chato.cl/teach

Instructor: Dr. Teodora Sandra Buda — https://tbuda.github.io/

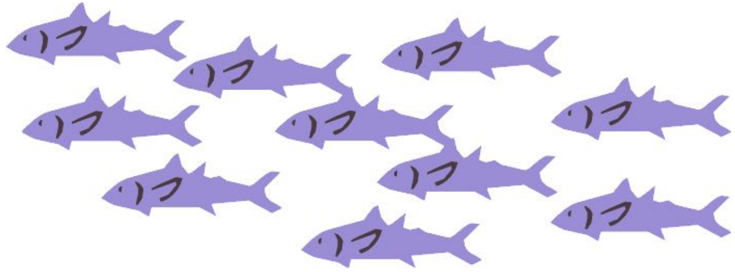
Sources

- Mining of Massive Datasets (2014) by Leskovec et al. (chapter 4)
  - Slides part 1, part 2
- Tutorial: Mining Massive Data Streams (2019) by Michael Hahsler

Probabilistic counting

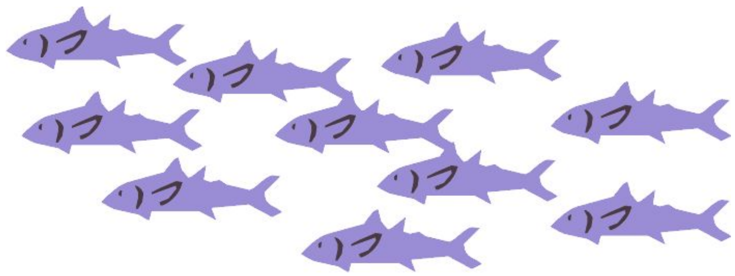
Counting fishes with pebbles

- Normally, to count with pebbles, you add one pebble every time you see an event
- How do you extend this method to count up to 1000 fishes with 10 pebbles?
- Assume you have access to a random number generator but not to an abacus for … reasons



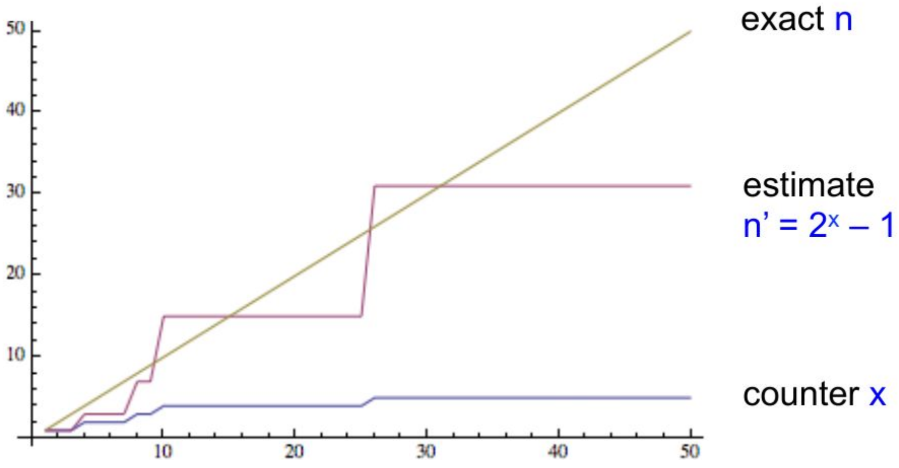
Answer

- How to count up to 1000 fishes with 10 pebbles?
- Every time you see a fish:
  - generate a random integer between 1 and 100
  - Add one pebble if that number is 1 (or any fixed value)
- Return *100 x number of pebbles* as an approximation



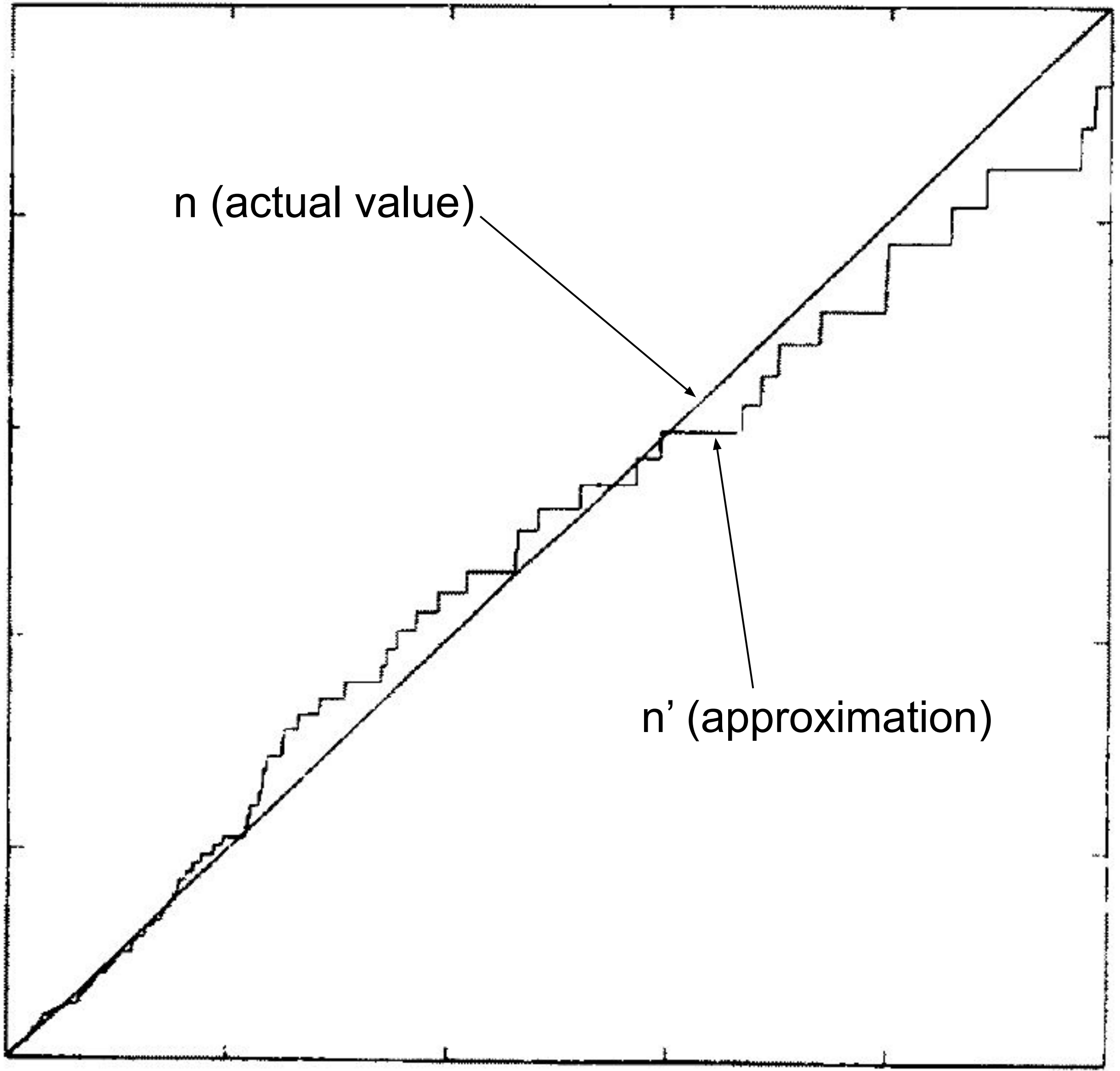
Morris' probabilistic counting (1977)

- $x \leftarrow 0$
- For each of the n events:
  - $x \leftarrow x + 1$ with probability $(1/2)^x$
- Return estimate $n' = 2^x+1$
- *Counter x needs only $\log_2(n)$ bits*



Source: [Slides by Nick Duffield]

- Simulation results by Flajolet (1985)



Morris' algorithm provides an unbiased estimator

- Init x=0, let $p_x = 2^{-x}$, estimate $n' = 2^x - 1$
- n = 1
  - before: $x = 0$ $p_0 = 1$;
  - prob. 1: $x \rightarrow 1$
  - estimate $n' = 2^1 - 1 = 1 = n$
- n = 2
  - before: $x = 1$; $p_1 = 1/2$
  - prob. ½: x stays at 1; $n' = 2^1 - 1 = 1$
  - prob. ½: $x \rightarrow 2$. $n' = 2^2 - 1 = 3$
  - $E[n'] = 1/2 \times 1 + 1/2 \times 3 = 2 = n$

Source: Slides by Nick Duffield

Morris' algorithm provides an unbiased estimator (cont.)

Let $X(n)$ denote random counter x after n^{th} arrival

Initialize $X(0) = 0$; increment w.p. $p_x = 2^{-x}$

Estimate $n' = 2^{X(n)} - 1$

$$
\begin{aligned}
E[2^{X(n)}] \quad & = \Sigma_{j=1,\ldots,n-1} \Pr[X(n-1) = j\,]\; E[2^{X(n)} \mid X(n-1) = j\,] \\
& = \Sigma_{j=1,\ldots,n-1} \Pr[X(n-1) = j\,]\; (\,p_j\, 2^{j+1} + (1- p_j)\, 2^j) \\
& = \Sigma_{j=1,\ldots,n-1} \Pr[X(n-1) = j\,]\; (2^j + 1) \\
& = E[2^{X(n-1)}\,] + 1
\end{aligned}
$$

Iterating: $E[2^{X(n)}] = E[2^{X(0)}] + n = 1 + n$

Therefore: $E[2^{X(n)} - 1] = n$

Source: [Slides by Nick Duffield]

Flajolet-Martin algorithm for distinct counting

Motivating example
how many neighbors?

- Let $n(u,h)$ be the number of nodes reachable through a path of length up to h from node u
- Naïve method
  - Maintain a set for each node u, initialize $S(u) = \{u\}$
  - Repeat h times:

$$S(u) = S(u) \cup \bigcup_{v \text{ neighbor of } u} S(v)$$

add to the set all the neighbours up to distance h

  - Answer $n(u,h) = |S(u)|$

not efficient to keep all the items, we just want the #elements

What is the problem with this method?

- Let *n(u,h)* be the number of nodes reachable through a path of length up to *h* from node *u*
- Naïve method
  - Maintain a set for each node *u*, initialize *S(u)* = *{u}*
  - Repeat h times:

$$S(u) = S(u) \cup \bigcup_{v \text{ neighbor of } u} S(v)$$

  - Answer *n(u,h)* = |*S(u)*|

Let’s look at each node

- We will receive a stream of items
  - Our neighbors at distance <= h
  - Repeated many times because of loops
- We want to use a small amount of memory
- We don’t care which items are in the stream
- We just want to know **how many are distinct**

Flajolet-Martin algorithm for counting distinct elements

- For every element u in the stream, compute hash $h(u)$

hash will be bit array

- Let $r(u)$ be the number of trailing zeros in hash value
  - Example: if $h(u) = 001011101\underline{000}$ then $r(u) = 3$

we look at the number of trailing 0s

- What is the probability of having $r(u)=1$? $r(u)=2$? $r(u)=3$?

r(u) = #number of trailing 0s

Flajolet-Martin algorithm for counting distinct elements

- For every element *u* in the stream, compute hash *h(u)*
- Let *r(u)* be the number of trailing zeros in hash value
  - Example: if *h(u)* = 001011101<u>000</u> then r(u) = 3
- Maintain $R = \max r(u)$ seen so far
- Output 2^R as an estimator of the number of distinct elements seen so far

Flajolet-Martin algorithm
(intuition)

- Let $r(u)$ be the number of trailing zeros in hash value, keep $R = max\ r(u)$, output 2^R as estimate
- Repeated items don't change our estimates because their hashes are equal
- About ½ of distinct items hash to *******0
  - To actually see a *******0, we expect to wait until seeing 2 distinct items
- About ¼ of distinct items hash to ******00

imp intuition

  - To actually see a ******00, we expect to wait until seeing 4 items
- …

USEFUL TO DRAW THE ELEMENTS

- If we actually saw a hash value of ***000...0 (having R trailing zeros) then on expectation we saw 2^R different items

Flajolet-Martin, correctness proof

- Let m be the number of distinct elements
- Let $z(r)$ be the probability of finding a tail of r zeroes
- We will prove that
  - $z(r) \to 1$ if $m \gg 2^r$
  - $z(r) \to 0$ if $m \ll 2^r$
- Hence 2^r should be around m

Flajolet-Martin, correctness proof (cont.)

- Probability a hash value ends in *r* zeroes = $(1/2)^r$
  - Assuming *h(u)* produces values at random
  - Prob. random binary ends in *r* zeroes = $(1/2)^r$
- Probability of seeing *m* distinct elements and NOT seeing a tail of r zeroes = $(1 - (½)^r)^m$

Imp to understand the proof

Flajolet-Martin, correctness proof (cont.)

- Probability of seeing m distinct elements and NOT seeing a tail of r zeroes = $(1 - (½)^r)^m$
- Remember $(1-\varepsilon)^{1/\varepsilon} \simeq 1/e$ for small ε
- Hence

$$\left(1-\left(\frac{1}{2}\right)^r\right)^m = \left(1-\left(\tfrac{1}{2}\right)^r\right)^{\frac{m\left(\frac{1}{2}\right)^r}{\left(\frac{1}{2}\right)^r}} \approx \left(\frac{1}{e}\right)^{\left(\frac{m}{2^r}\right)}$$

Flajolet-Martin, correctness proof (cont.)

- Probability of seeing m distinct elements and NOT seeing a tail of r zeroes $\approx (1/e)^{\left(\frac{m}{2^r}\right)}$
- If $m \gg 2^r$, this tends to 0
  - We almost certainly will see a tail of r zeroes
- If $m \ll 2^r$, this tends to 1
  - We almost certainly will not see a tail of r zeroes
- Hence, 2^r should be around m

Flajolet-Martin: increasing precision

- Idea: repeat many times or compute in parallel for multiple hash functions
- How to combine?
  - Average? $E[2^r]$ is infinite, extreme values will skew the number excessively
  - Median? 2^r is always a power of 2
- Solution: group hash functions, take median of values obtained in each group, then average across groups

Let's go back to counting neighbors

Naïve method:

Maintain a set for each node *u*, initialize *S(u)* = *{u}*

Repeat h times: $S(u) = S(u) \cup \bigcup_{v \text{ neighbor of } u} S(v)$

Answer *n(u,h)* = |*S(u)*|

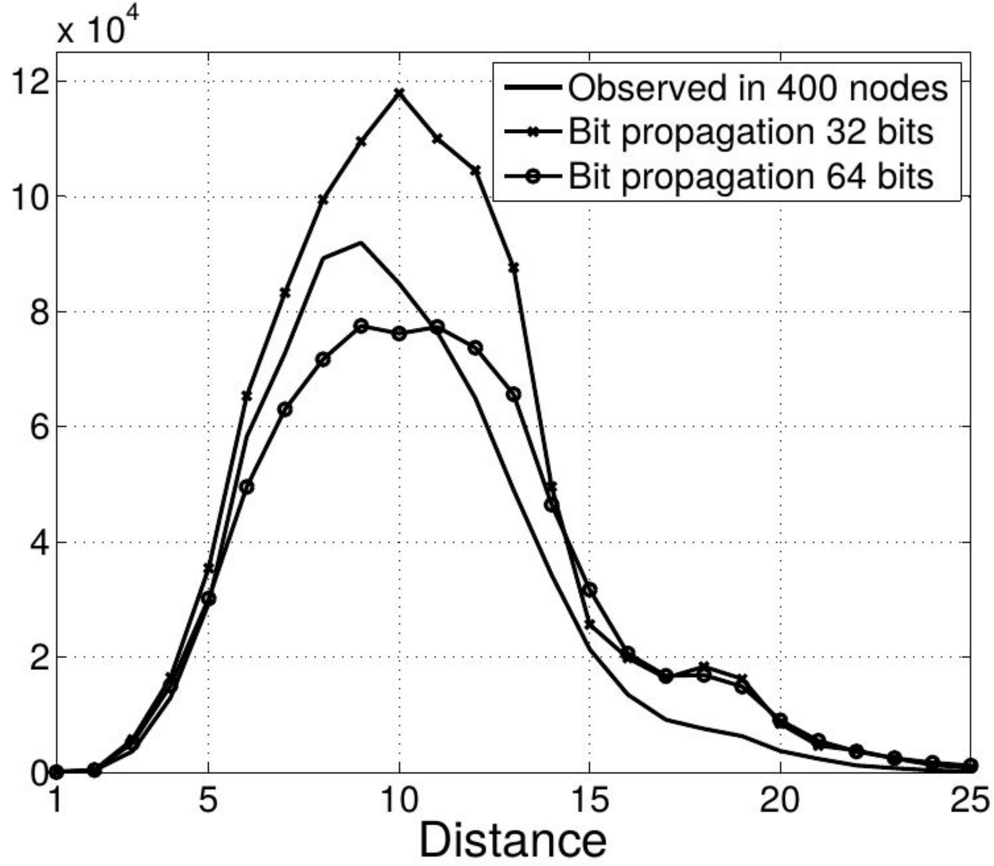
ANF method:

// Set $\mathcal{M}(x, 0) = \{x\}$
FOR each node x DO
$M(x, 0) =$ concatenation of k bitmasks each with 1 bit set ($P(\text{bit } i) = .5^{i+1}$)
FOR each distance h starting with 1 DO
FOR each node x DO $M(x, h) = M(x, h-1)$
// Update $\mathcal{M}(x, h)$ by adding one step
FOR each edge (x, y) DO
$M(x, h) = (M(x, h)$ BITWISE-OR $M(y, h-1))$
// Compute the estimates for this h
FOR each node x DO
Individual estimate $\hat{IN}(x, h) = (2^b)/.77351$
where b is the average position of the least zero bits in the k bitmasks

Palmer, C. R., Gibbons, P. B., & Faloutsos, C. (2002, July). ANF: A fast and scalable tool for data mining in massive graphs. In Proc. KDD.

Example of another variant of the same type of algorithm

- More repetitions of the algorithm yield better precision



Becchetti, Luca, Carlos Castillo, Debora Donato, Stefano Leonardi, and Ricardo Baeza-Yates. "Using rank propagation and probabilistic counting for link-based spam detection." In Proc. of WebKDD, 2006.

Summary

Things to remember

- Probabilistic counting algorithms:
  - Morris
  - Flajolet-Martin

Exercises for TT22-T26

- Mining of Massive Datasets (2014) by Leskovec et al.
  - Exercises 4.2.5
  - Exercises 4.3.4
  - Exercises 4.4.5
  - Exercises 4.5.6